Convolutional Neural Networks for Object Recognition

Alistair Grom - 964398 - CSC345

December 11, 2020

Introduction

The CIFAR-10 Data set is commonly used in training machine learning algorithms. It contains 60000 images in 10 classes. These classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. We are using a subset of this data set which only contains 1000 images per class. Object Recognition and Image Classification are large areas of Machine Learning. In this case we have a supervised learning problem, which is learning based on a given data set with labeled data that provides us a schema for labeling other similar data.

We are given 4 data sets; trnImage, a set of 10000 32x32x3 training images, trnLabel, a set of 10000 training labels tstImage, a set of 1000 32x32x3 testing images. tstLabel, a set of 1000 testing labels.

The task is to devise a model that can learn from the trn sets to be compared for accuracy with validation data with at least some accuracy (higher than random).

My proposed approach to classifying this data set was through a convolutional neural network (CNN). Through my research into the CIFAR data set I could see a large amount of content based on CNNs for in relation to this data set. The tensorflow tutorial on CNNs uses the CIFAR data set in their example [1]. The website Benchmarks.AI lists variations of CNN and Deep Learning near the top [2]. Alongside high accuracy ratings that CNNs are able to achieve in this task there is also a lot of support on the internet to help with improving and optimising models. Another reason for this choice was that CNNs do not require a large amount of data preparation before handing the data to the training model.

The model I have created demonstrates the ability the CNNs have the ability to accurately recognise objects given to it via a data set. With a %62.4 validation accuracy and 1.168 loss on the subset of CIFAR-10 provided.

Method

Data Preparation

First I needed to gather the data from the files given and reshape this data to match that of what the model I am using expects by transposing the data format. A decision I made was to create my validation data/testing Data from the training data provided as opposed to using the data set provided. I chose this as this was similar to how I have approached CNNs in the labs also this gave me double the validation data size, 2000 compared to 1000, while only reducing the training data size by 2000 to 8000. This did not appear to have much impact on accuracy in my experiment.

My approach does not use a dedicated feature extraction such as the HOG method provided but a Deep Learning approach where the feature extraction

and classification takes place in one model. [3]

Convolutional Neural Network Model

I am using is a sequential model as we have data with one input tensor and an output tensor for each image [4]. Through my model I use various layers of feature extraction.

Conv2D - Feature Extraction

A convolutional filter is used at the start of the model to extract simple features of an image. We can use multiple of these to get higher level features. In my model I start off by using a Conv2D with 32 filters of 3x3 with ReLU activation. This method provides a computationally fast way of extracting features. This layer is used twice more in my model. With 64 filters and 128 respectively. When testing with only one the validation accuracy stopped around 50% whereas the model was able to grow past 60% with one or two implemented.

MaxPooling2D - Dimensionality Reduction

The purpose of max pooling is to "accumulate" features from maps generated by convolving a filter over an image [5]. The case of max pooling involves taking the largest value from a sample and reducing that portion of the image to that value. This helps over-fitting and reduces the computational load of the model.

In my model I use max pooling with a 2x2 filter. I use this after two uses of the convolutional filters. This in effect halves or more than halves the size of the image we are learning from.

Dropout Dropout is a method of combating over fitting. This is done by randomly deleting some of the connected nodes in the network during training [6]. Over fitting can be the result of overcomplex networks that in some way get away from the original task during the learning procedure. This method possibly is not the best method but for the purpose of this task speed and efficiency are important due to hardware and relative size of the data set. In my model I use a dropout of 0.3, this value allows the accuracy to still remain the same but reduces the validation loss after the filtering is done. This has reduced the validation loss and keeps it around 1.1 as opposed to 1.8 after around 20 epochs.

Flatten This layer involves flattening the pooled feature map allowing us to insert this data into the neural network, so we can process it.

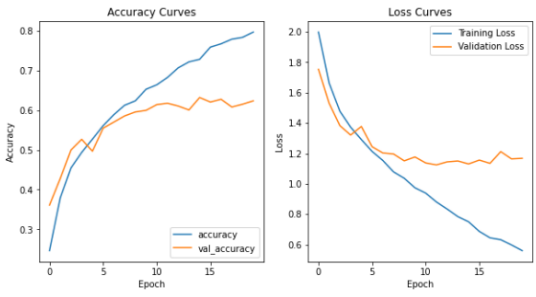
Dense Dense layers take vectors as input. In my model the data that is passed to this layer is flattened meaning it is 1D. First I add a dense layer with 128 outputs to reduce the data from a 1D array of 512 items to an array of 128, then again with a value of 10 to represent the 10 classes we are classifying our data into.

Compilation When compiling a CNN the model has many different arguments that massively change the use case and performance of the CNN depending on the problem.

I have chosen the Adam optimiser. This is a straight forward optimiser that can perform well without complex fine tuning and works well without advanced hardware. In addition to this, this method is used in the tensorflow document about CNNs [1]. The loss function used is Sparse Categorical Cross Entropy. We have two label classes meaning this is a good function for this task. The last component of the compilation is the metric we are looking at, in this case we are looking at accuracy of the validation data compared to the true values. I am using 20 epochs with 250 steps per epoch, the accuracy increase is negligible beyond 25 and the validation loss can start to rise slightly.

Results

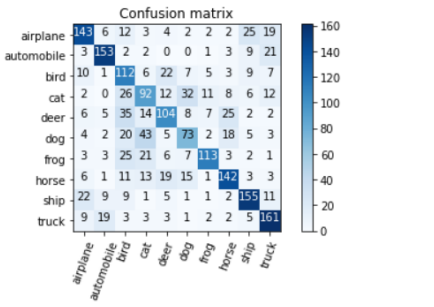
This model resulted in %62.4 validation accuracy and 1.168 loss across all 10 categories. We are using accuracy as the metric of success for this model, also considering loss but accuracy is the main goal.



(a) Accuracy and Loss

Classification report

	precision	recall	f1-score	support
0	0.69	0.66	0.67	218
1	0.77	0.79	0.78	194
2	0.44	0.62	0.51	182
3	0.46	0.46	0.46	201
4	0.58	0.50	0.54	208
5	0.50	0.42	0.45	175
6	0.78	0.61	0.69	184
7	0.68	0.66	0.67	214
8	0.70	0.72	0.71	216
9	0.67	0.77	0.72	208
accuracy			0.62	2000
macro avg	0.63	0.62	0.62	2000
weighted avg	0.63	0.62	0.62	2000



(b) Classification report and confusion matrix

We see where the model starts to over fit around the 10 epoch mark. After introduction of the Dropout layer the line says horizontal rather than increasing

the loss. This is also a similar point to where the model stops learning at as fast a pace.

The confusion matrix shows a success of the model. All the 'correct' categories that have been learned have matched up most with themselves (the highest confusion value is that of itself in each).

Another metric of success is that many of the more distinct categories have a very dominant accuracy, above %77 in automobile and truck.

We see that the categories that have the lowest % of accuracy are that of similar looking animals. As expected cat and dog are the most mistaken two groups.

Conclusion

The approach of using a Convolutional Neural Network for object recognition is clearly one of the most successful uses of this area of machine learning. With people able to achieve upwards of 90% on the CIFAR data set [2]. While I was unable to reproduce accuracy of that level I was able to create a model that can learn from this data given and also the full CIFAR-10 data set also (to greater success $\tilde{7}0\%$). On this given data set.

With more research into this topic I can definitely see where these CNNs can be improved. From more advanced data preparation and much more complexed layers. I noticed that some areas in this data set maybe are not enough to train a model, such as only having around 200 dogs and cats a model may struggle to learn. This could be fixed with a larger data set so we can extract more distinct features further separating these categories. I also noticed that my model did reach a point of diminishing returns when running for more epochs with more steps. This can be evidenced by the slow down in growth of the accuracy curve and the need for me to use a dropout layer. This may suggest the limitations of my approach are near this boundary.

Bibliography

[1] Tensor Flow, Google, 'Convolutional Neural Network (CNN)', $https://www.tensorflow.org/tutorials/images/cnn$

[2] Benchmark.AI, $https://benchmarks.ai/cifar-10$

[3] Spectral–Spatial Feature Extraction for Hyperspectral Image Classification: A Dimension Reduction and Deep Learning Approach, IEEE Transactions on Geoscience and Remote Sensing Zhao, Wenzhi and Du, Shihong, 2016, vol=54

[4] keras, 'The Sequential Model', $https://keras.io/guides/sequential_model$

[5] DeepAI, 'Max Pooling', $https://deepai.org/machine-learning-glossary-and-terms/max-pooling$

[6] CSC345, 'Lecture 10 - CNN', Over fitting, Slide 39, Swansea University, Prof. Xianghua Xie